Smart Home Clima

Software Engineering for Autonomous Systems

A.A 2022-2023

Student
Davide Romano

Index

Source code: https://github.com/daviRoma/seas-smarthomeclima

1. Summary

The developed application represents an autonomous system of a generic smart home.

An autonomous system combines sensors and control systems in such a way as to enable complex sequences of operations that can be performed on different types of systems. The need to develop systems of this type arises from the need to be able to manage increasingly complex and constantly evolving systems.
Autonomous behaviour begins with the creation of a plan, shaped to achieve one or more goals subject to a certain set of constraints and resources. In general, it consists of automatic control and definition of a sequence of operations which, if necessary, can be changed.
The advantage of this type of application lies mainly in the possibility of operating in hostile environments without the aid of man or which require constant monitoring over time.

There are various levels of automation of a system:

- Controlled by user
- Assisted by user
- Delegated by user
- Supervised by user
- Hybrid
- Completely autonomous

Despite the numerous advantages and benefits of such systems, there are various 'uncertainties' during their development which have not allowed their large-scale diffusion to date. For example, the danger associated with their possible failure.
Just think of autonomous automotive systems which, due to incorrect or poorly implemented choices, could lead both to damage to the element managed by the system and to the loss of human lives.

In the next chapter will be introduced the main concepts of MAPE-L Loop, the architecture implemented for the system described in this document. Chapter 3 describes assumptions and constraints declared for the project, chapter 4 describes the objectives to be achieved using this application, focusing on the main requirements. In chapter 5 and 6 instead the technologies and devices used. The implemented functions are described in chapter 7, while in chapter 8 we can find the static view and a dynamic view of the software. Chapter 9 describes the graphical interface implemented and finally, chapter 10 and 11 describes the results obtained from the running system.

2. MAPE-K Loop

A system is defined autonomous if it respects the MAPE-K Loop, which describes the flow of data between the various components of the system itself.

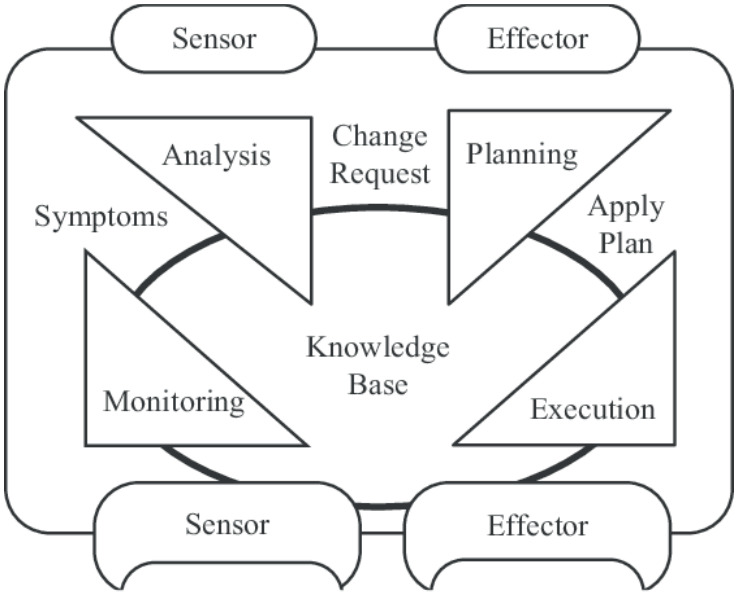

Figure 1 - MAPE-K Loop

- **Monitoring**: Component that takes care of capturing the properties of the environment, physical or virtual, that are important for the purposes of system automation (i.e. collects data from the managed resource, provides mechanisms to aggregate and filter the data flow in entry, store relevant and critical data in the knowledge base for future reference).

- **Analysis**: Component that compares event data with knowledge base models to diagnose symptoms and store them, but mostly correlates incoming data with historical data, analyzes symptoms and predicts problems.

- **Planning**: Component that considers the data monitored by the sensors to produce a series of changes to be made on the managed element. Interpret symptoms to come up with a plan, decide on an action plan, construct actions, and implement policies.

- **Execution**: Component that executes the change of the process managed by the actuators and executes the plan.

- **Knowledge**: The four components of a MAPE-K cycle work together through the exchange of knowledge which is represented by this component.

The implemented system was developed in such a way as to respect all the aspects described above and to be applied in a specific context, the smart house. The first necessary requirement is therefore to have sensors

and actuators, respectively able to collect data and to make changes through the activation or deactivation of physical components.

A data simulator has been developed to reproduce a real climate situation of a house. In particular, the trend of the following factors was simulated: internal temperature, conditioned by the external but not monitored temperature and by internal insulation, movement inside a room.

The advantages of building such a system lie in the possibility of automating human actions, in such a way as to allow the system to work optimally according to the set mode and save energy, also avoiding errors deriving from distractions.

3. Assumptions & Constraints

The implemented system was developed respecting some assumptions and constraints to limit the context and make it stable and reliable. Of course, the implementation is open to some extensions.

1. The system was limited to a single smart home:
    a. a single user can manage only one smart home,
    b. the smart home entity was not modelled.
2. The executor component updates the actuators only for testing and simulation purpose. In an IoT system the executor only sends inputs via MQTT to the actuators.
3. The domain entities are part of the Kwnowledge.
4. It was decided to separate the system to manage only two seasons: winter and summer.

4. Scope

Often there are unused rooms at certain times of the day in a house, for this reason it makes no sense, for example, to keep a high temperature (winter) in all the rooms, but it would be enough to keep it on a certain threshold that is easy to reach in case someone is inside or when that room will be used at a certain time.

We want to ensure that the optimal temperature chosen by the user is always maintained in all rooms and we think that this type of management can also help save energy.

For each smart room there is a:

- **Optimal value**
- **Predictive margin**
- **Danger margin**

in such a way as to immediately identify the situation in which the room is located and, if necessary, activate a plan.

In the following figure are showed reactive range of the system.

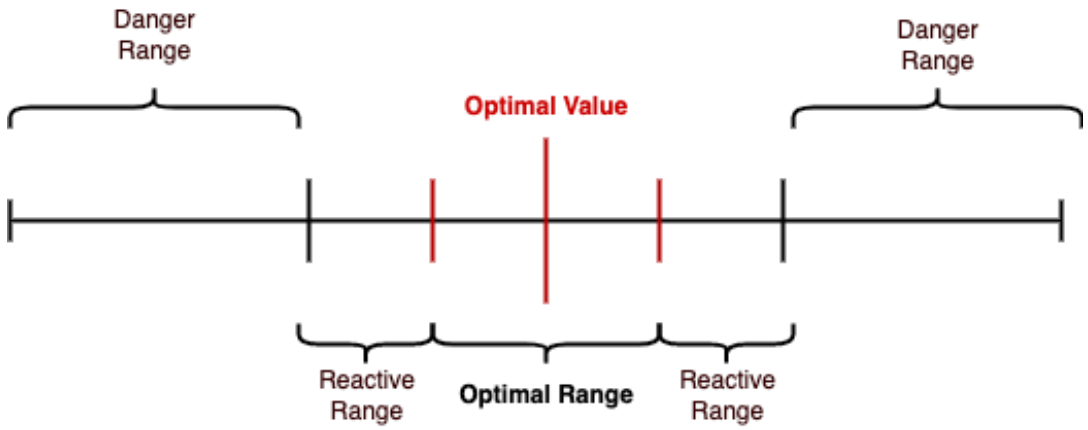

Figure 2 - Reactive Range

The optimal range is the one in which the actuators should not be active because the temperature is near the optimal value.
In the predictive range the temperature has dropped below the optimal threshold, therefor the system has to activating a plan to reach the optimal value for that room.
The last one, the danger range, is the threshold in which the system must activate a plan and has also to consider a mode change to deliver more power and reach the next threshold sooner.

The mode will be changed only if the temperature will fall below a certain threshold, calculated like this: *OptimalValue - OptimalMargin - DangerMargin - 1* (winter example). A pianification will be created with mode set up to **POWER** and the **power value** set to maximum value.

5. Devices

In this section are described the devices used from our system.

These are divided in Sensors and Actuators. Sensors detect the changes of a certain state inside a smart room, actuators perform the actions to maintain an optimal state within the room.

5.1 Sensors

Name	Description
Temperature	Temperature detection sensor inside a smart room. One for each room.
Motion	Motion detection sensor inside a smart room. One for each room.
Contact	Contact detection sensor for each window inside a smart room.

Table 1: Sensors

5.2 Actuators

Name	Description
Radiator	Physical object able to increase the temperature of the smart room. It is assumed that there is at least one per room.
Conditioner	Physical object able to decrease the temperature of the smart room. It is assumed that it may not even be present in all rooms
Window	The window of a room. One or more per smart room.

Table 2: Actuators

6. Technologies

The software has been completely implemented through **Java** and **Angular2+**. The following technologies are used to complete the entire system functionality:

- **Mosquitto**: Message broker implementing the MQTT protocol.
- **Eclipse Paho**: Eclipse module that provides an open-source client implementation of MQTT messaging protocols.
- **Spring framework**
- **H2**: SQL in memory database.

In a particular way, Angular2+ was mainly used to develop a graphical interface and for interacting with the system which expose APIs with Spring Boot.

The Java project was developed using the module dependency pattern using **Maven**. The root project has the *pom.xml* containing all the project module, while each module's *pom.xml* contains the dependency to the other module.

7. System Features

The **SmartHomeClima** system has as its main objective that of guaranteeing an optimal temperature for all the rooms.

In order to reach this objective, the system has some "reactive functions" which allow the system to react to non-optimal situations that can arise inside the smart home during the day.

7.1 Reactive

Reactive functions are executed when a plan is triggered, as the plan is defined when a non-optimal situation is detected.

Reactive functions are:

- activate a Radiator/Conditioner when a reactive or danger scenarios was detected, and a plan is activated,
- close the window if a danger scenario was detected and a plan is activated,
- activate a Radiator/Conditioner when the motion sensor detects a movement, and the smart room has not the optimal temperature.

7.2 Modes

These modes consist of the energy settings we have available. They are set by the user through the graphical interface, but they can also be set by the system (only for danger mode). All the applicable modes and their characteristics are described below.

- **ECO**
  It consists of the energy saving plan, as well as the default mode of the application. With the ECO mode active we limit the power of the actuators such as conditioners and radiators to a single value, 1.

- **COMFORT**
  With the Normal mode active, we limit the power of the conditioners and radiators in a two value range: 1 and 2. This mode, in a real environment, is on average more expensive than the ECO mode in terms of consumption, but at the same time it manages to bring the system back into optimal range faster and more efficiently.

- **POWER**
  This mode is the maximum that can be set in the system from an energy point of view and potentially the most expensive, favouring execution speed over consumption. In this mode, in addition to the reactive functions, the system can make the most of the power of the conditioner and radiators, in particular it can set them to powers such as 2, 3, 4.

Each mode has a predefined and not editable duration, for example a pianification with the ECO mode active takes 20 minutes, after that time the pianification will be deactivated if the optimal temperature range has been achieved.

7.3 Seasons

The system can be configurated for two different seasons: Winter and Summer. This kind of difference was adopted because these are the two seasons in which air conditioners and/or radiators are mostly used.

In our system we want to leave a little more freedom to the user for configuration.

However, one could think of removing this difference and extending the system with external temperature detection sensors and adopting changes based on the time of year.

- **WINTER**
  When the policy group is set to *Winter*, the system know that it has to activate only the radiators and not the conditioner when a reactive or danger scenario will be verified.

- **SUMMER**
  On the other end, if the policy group is set to *Summer*, the system know that it has to activate only the conditioners.

7.4 Configurator

The **configurator** is a support entity of the system. It is not stored in the database, and it contains the status codes of the smart room, like: code for optimal temperature reached, code for danger case, mapping *mode* to available power to set, and so on.

7.5 MQTT Channels

The MQTT channels are used by the system to communicate with sensors, actuators, and the control panel.

- **SENSOR CHANNELS**
  This is the channel used by the sensors to send the detected data. The Monitor component is subscribed to the channel, receives all the data sent by the sensors and then it will process them.

- **ACTUATOR CHANNELS**
  This is the channel used by the Executor to send the action data to the actuator. Each actuator is subscribed to the own channel and receives the action data to apply.

- **MONITOR ACTUATOR CHANNELS**
  The actuator channels are used by the Control Panel to receive the same data sent to the actuators by the Executor.

- **MONITOR EXECUTOR CHANNELS**
  The executor channels are used by the simulator to receive the data sent to the actuators and simulate the actuators behaviour.

8. System Architecture

In this chapter will be represented a static view of the system using the Component Diagram and the ER-Diagram.

8.1 Component Diagram

A component diagram describes the organization and wiring of the physical components in a system.

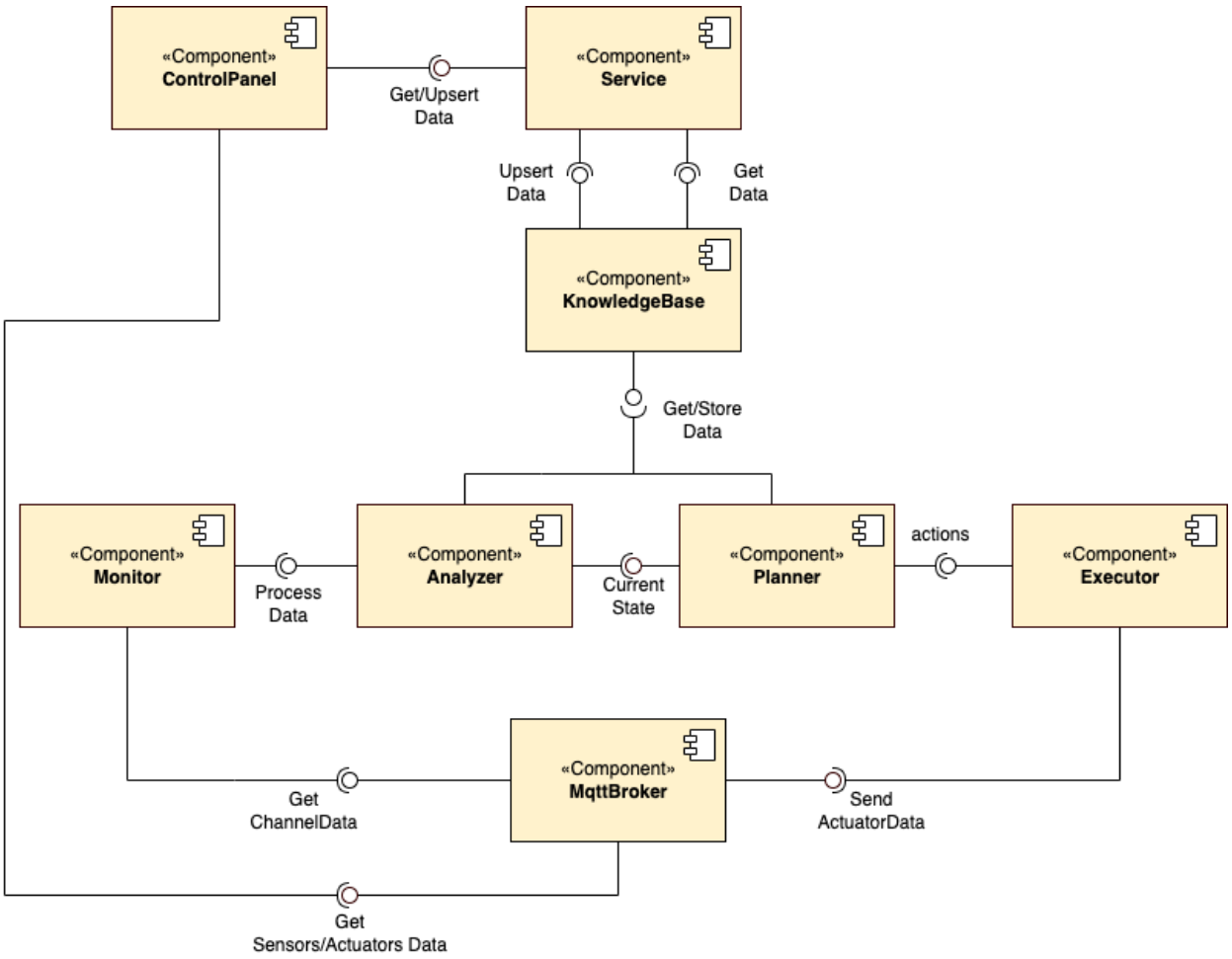

Figure 3 - Component Diagram

The **simulator** has been included for simplicity in the **Service** component and because it can be activated only from an API interface called by the **ControlPanel** component. For this reason and because in a real system the simulator is not inserted, it was not represented in the Component Diagram.
The architectural style used to create the system is based on the MAPE-K loop, consequently the main components are represented by **Monitor**, **Analyzer**, **Planner**, **Executor** and **KnowledgeBase**.
The **Monitor** will be responsible for collecting data (by subscribing to the data reception channels) and sending them to the **Analyzer** component, which will take care of monitoring any situations of reactivity or "danger" and the subsequent sending of the results products to the **Planner**. The latter component takes care of preparing appropriate actions based on the situation explained by the Analyzer.
At the end of this flow, the **Executor** component, once the information received from the Planner has been collected, implements any expected status changes. It also takes care of updating the graphical interface by publishing all status updates on the appropriate MQTT channels. All the data relating to the current situations of each room are shown in real time on the **ControlPanel** graphical interface. In addition to viewing, a user

can interact with the system itself and make the desired changes. Finally, the **KnowledgeBase** component represents the component that interfaces with the MySQL database and allows access to the predictive functions of the system.

8.2 ER Diagram

The ER-Diagram represents the data-model of the SmartHomeClima system.

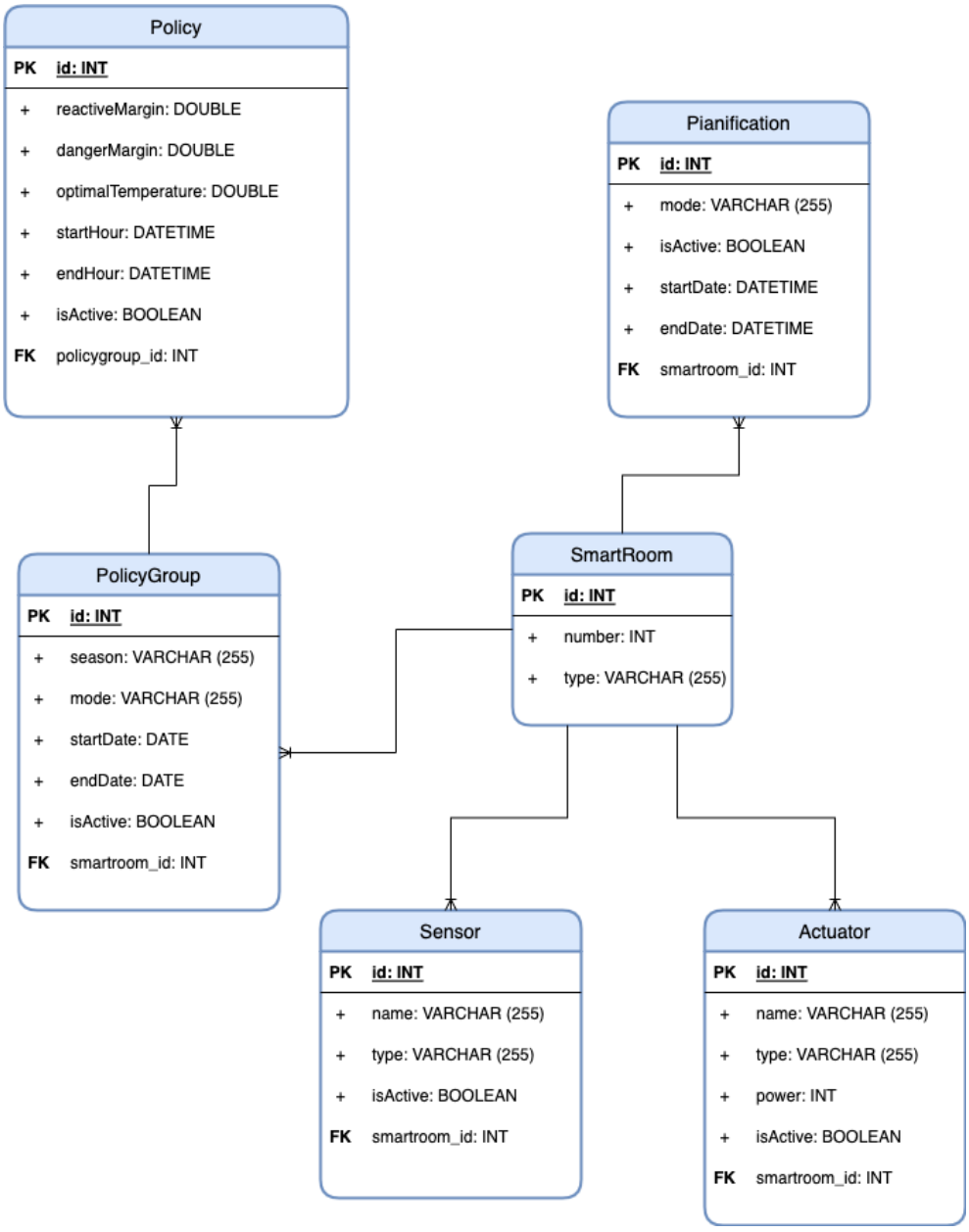

Figure 4 - ER - Diagram

The main entity of the data-model is the **SmartRoom** and corresponds to a single room in an house. As mentioned in the Assumption & Constraints section, the *SmartHome* entity was not represented because it was intended to represent the instance of a single house. The number attribute represents a symbolic number to assign a room, and the *type* attribute distinguish the type of the room, like *bedroom*, *living room*, *bathroom*, etc.

The **Sensor** and the **Actuator** entities represent the sensors and the actuators installed in a room respectively. Each room must have at least one sensor and at least one related actuator, otherwise it wouldn't make sense to call the room "smart" and register it in the database. The two entities have the *type* attribute which assumes one of the value showed in the Table 1 and 2 descripted in the section 5.
The **PolicyGroup** entity is the smart room scheduler and a **Policy** container. It contains attributes like *startDate* and *endDate* which describes the duration of the scheduler. That scheduler has the *mode* and *season* attributes described in the section 7.2 e 7.3. The system can have multiple policy groups but only one per time can be active.
The **Policy** describes the state of the room (optimal temperature and reactive ranges) at certain time with the help of attributes *startHour* and *endHour*.
At the end we have the **Pianification** entity. Such model represents a single pianification that is being adopted for a room after a situation of reactivity or danger has been detected. The *mode* attribute is also present here because in case of danger scenario the mode can be changed and a major value of power for the actuators can be set.

8.3 Sequence Diagram

This section shows the dynamic view of the running system through the use of the Sequence Diagram. In particular, a high-level execution of the system is reported in order to highlight the various interactions between the components within the application.

8.3.1 System execution

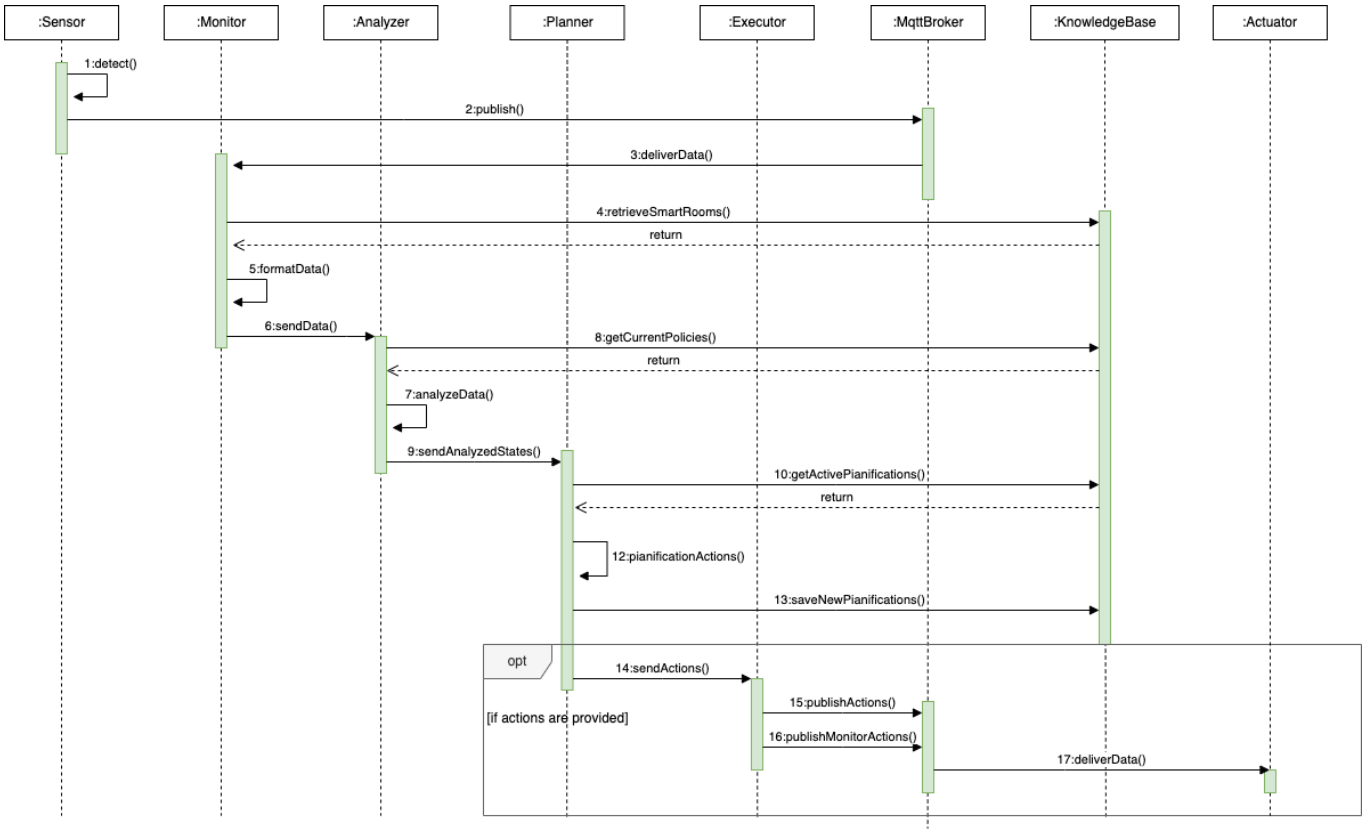

Figure 5 - Sequence diagram which showing the standard execution of the system.

This scenario describes the flow of data and the behavior of the system in its standard execution. The sensor and actuator components have been inserted in order to show the complete execution and to make the flow clearer. Initially, the sensor collects some information and sends it to the MQTT broker. Subsequently, through the MQTT channels, it sends the information in its possession to the Monitor which, after an appropriate formatting and grouping phase, will take care of sending everything to the Analyzer. The latter will collect the data relating to the rooms by interfacing with the **KnowledgeBase** obtaining from it the active policies and the status codes from the configuration to describe the current situation. At this point the Analyzer will transfer control to the Planner who, after having obtained the floors active at that instant, will take care of carrying out any actions necessary to maintain the optimality of each room, using the status codes received previously. If actions are produced, they will be sent to the Executor who will take care of forwarding them to the broker and to the actuators who will then have to physically execute the action.

8.3.2 Show data and update by the user

The following scenario shows the behavior of the system when a user interacts with it.

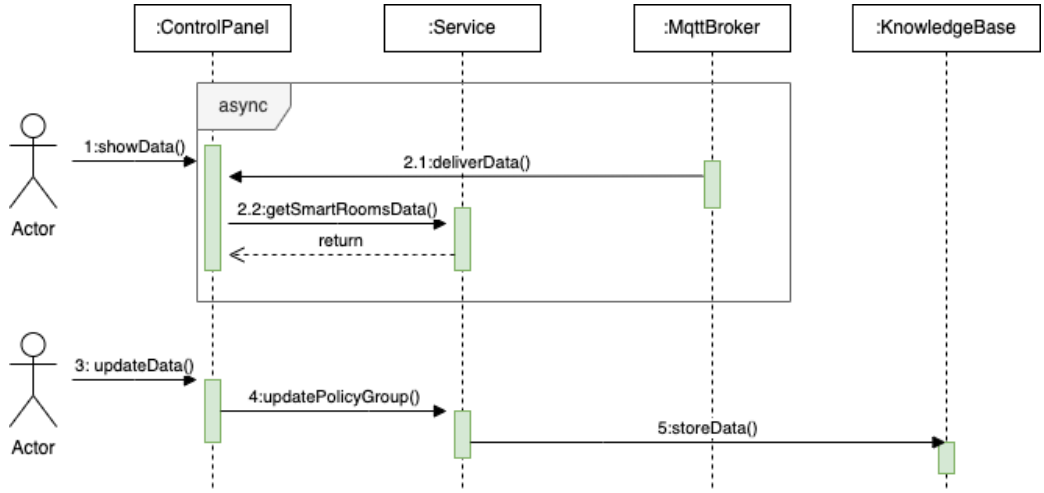

Figure 6 - Sequence Diagram which showing user request for data and update.

In this scenario the user opens the Control Panel interface and request to show the smart rooms data. This is an asynchronous operation because the smart rooms data and the actuators and sensors date arrived from two different sources.
In the second scenario the same user request for update an entity, the Policy Group in this case. The request is forwarded to the Service component through the Rest API, then the Service send the request to the KnowledgeBase component which stores the new information.

9. Control Panel UI

The user interface was implemented using Angular2+ which with the MQTT module was able to both receive and publish data to the broker. The below figures show the implemented graphic interface.

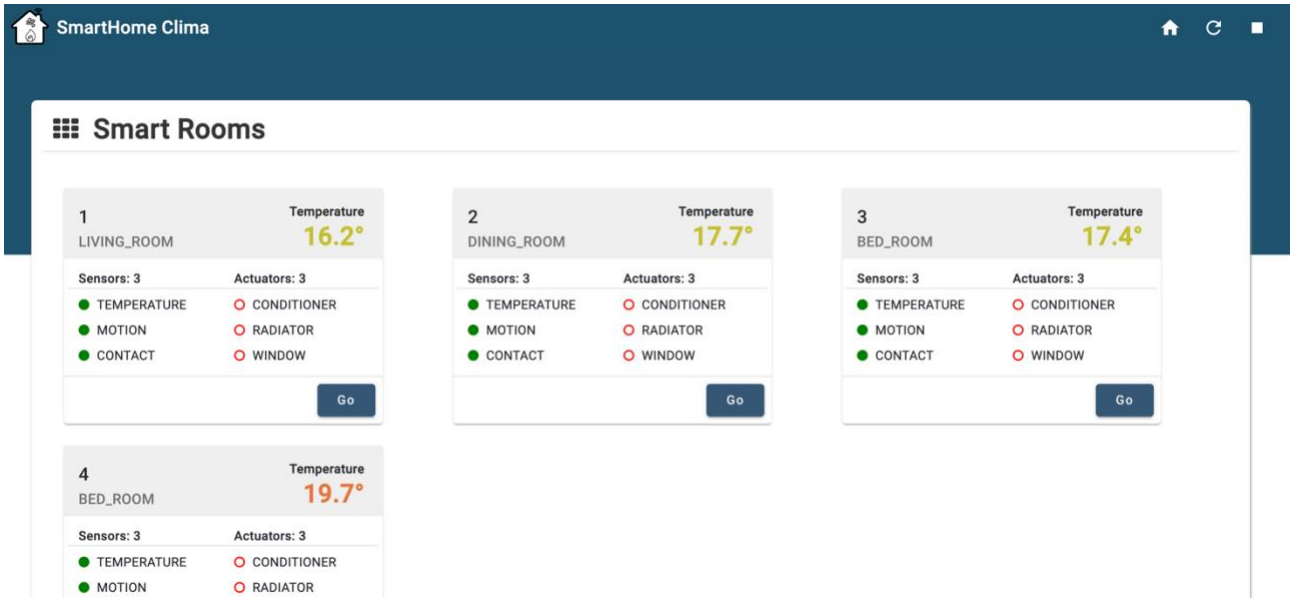

Figure 7 - ControlPanel home page

The **ControlPanel** page is the main page of the web application, it shows all the smart rooms and its states. In the top-right of the page we can find the button which starts the monitoring of the smart home (MAPE-K loop).

Each card represents a single smart room state in which we can find:

- number and the type of the room on the top-left,
- temperature on the top-right,
- sensors and actuators states: green icon for active and red empty icon for inactive stat,
- Go button to navigate to the smart room detail.

On the bottom right there is the “Add” button which allows the user to register a new smart room.

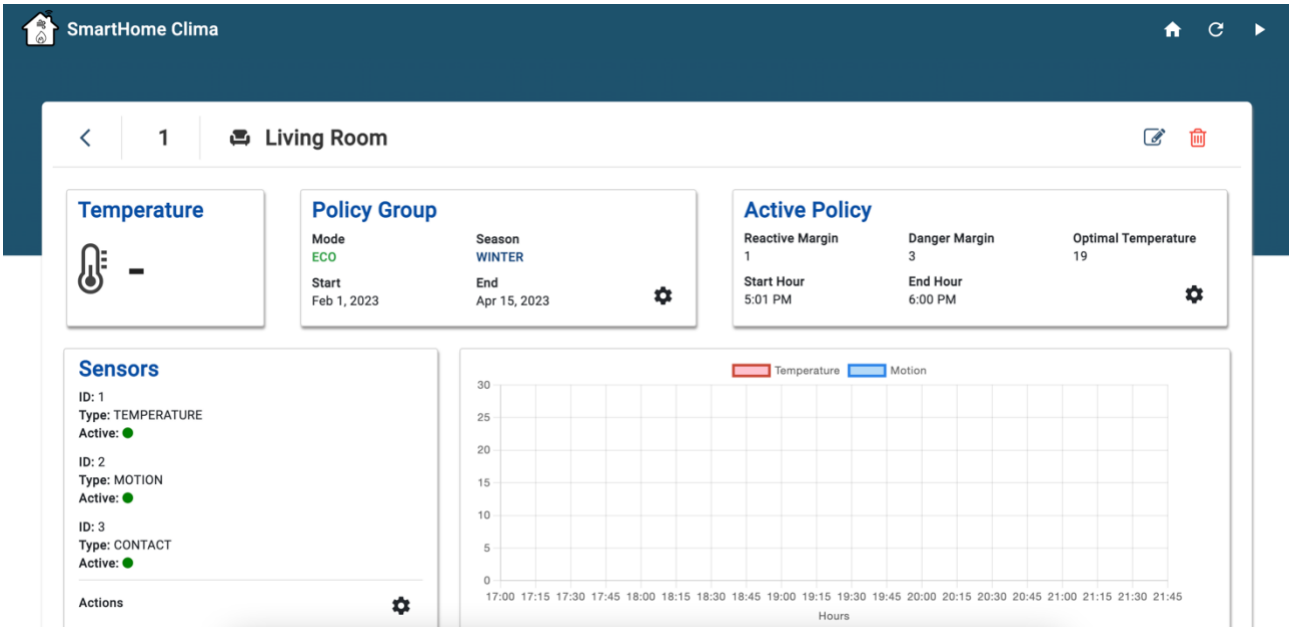

Figure 8 - Smar Room detail page

The above figure shows the smart rooms details and state. The Policy Group card contains the current active mode and season, while the Policy card contains the current active Policy of the system.
In the central part of the page, we find the sensors and actuators registered in that smart room and the graphs that define their progress.
The gear icon present in each card allows the user to change the database information concerning the corresponding entity (edit, create, update, and delete).

10. Execution Example

In this chapter will be shown an example of the running system. In particular, an execution of the system is set in *winter*.

The Simulator component generates a random value for each sensor and publish the value to the corresponding Mqtt channel, then pauses itself and restarts every 30 seconds generating sensors data by following this logic:

- Temperature sensor:
  - if a Pianification is currently active for the smart room, then add a random number between 0 and 1 to the temperature value,
  - if no Pianification is currently active for the smart room, then subtract a random number between 0 and 1 to the temperature value,
  - publish data to the corresponding Mqtt channel.
- Motion sensor:
  - generate a random value 0 or 1,
  - publish data to the corresponding Mqtt channel.

At the end of the loop, receives the messages from the executors and updates the actuators data on the database.

In this particular example, we focused on a single smart room to analyze how the temperature changes based on the logic described above in a range of 3 hours.

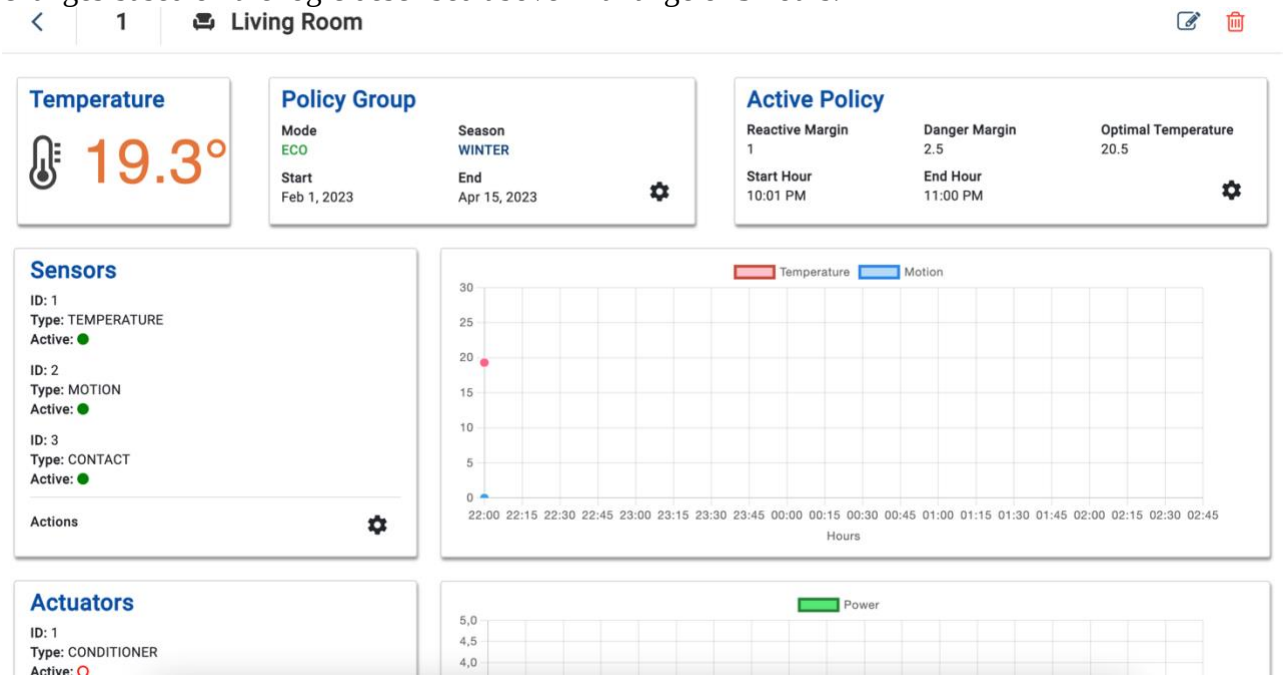

Figure 9 - Smart Room initial state

The figure 9 shows the initial state of the smart room. Sensors and actuators send their first detected value and the Policy card shows what is the optimal value to reach for that room.

In the next figure we can see how the system reacts to the temperature or motion value changes. The first graph is the sensor graph which shows the values detected by the temperature and motion sensors. The second graph shows the values assumed by the actuator (conditioner or radiator).
As we can see, every time the temperature sensor detects a decreasing temperature value or the motion sensor detects a movement in the room, the radiator is turned on. This behavior is provided by the planner, it activates a pianification for the room based on the mode indicated from the policy group.

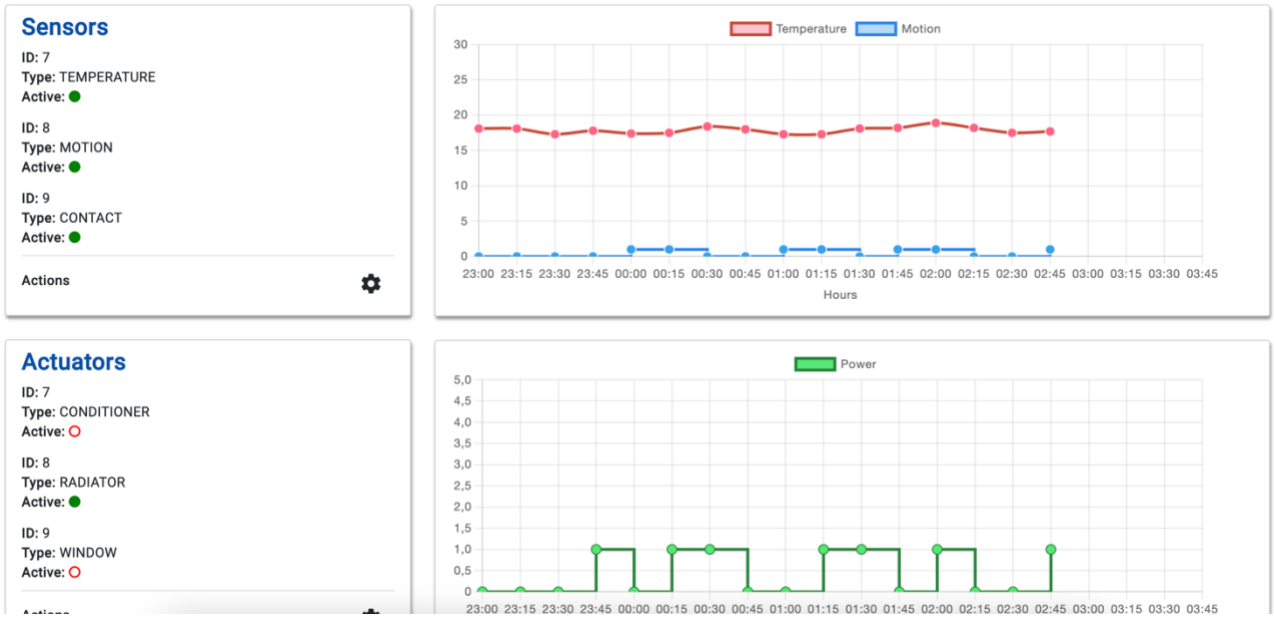

Figure 10 - ControlPanel: BedRoom monitoring

The last figure only shows the application logs in which we can find the messages sent to monitor the situation from the Control Panel and the logs relatives to the analyzer and planner components.

```
[       Thread-3] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::[publish]::Send message to smartroom/2/monitor/actuator/5
[564688730583875] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::The delivery has been complete. The delivery token is 1
[       Thread-3] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::[publish]::Send message to smartroom/2/monitor/actuator/6
[564688966458709] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::The delivery has been complete. The delivery token is 1
[       Thread-3] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::[publish]::Send message to smartroom/3/monitor/actuator/7
[564689203098292] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::The delivery has been complete. The delivery token is 1
[       Thread-3] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::[publish]::Send message to smartroom/3/monitor/actuator/8
[564689439603375] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::The delivery has been complete. The delivery token is 1
[       Thread-3] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::[publish]::Send message to smartroom/3/monitor/actuator/9
[564689676126667] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::The delivery has been complete. The delivery token is 1
[       Thread-3] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::[publish]::Send message to smartroom/4/monitor/actuator/10
[564689914700584] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::The delivery has been complete. The delivery token is 1
[       Thread-3] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::[publish]::Send message to smartroom/4/monitor/actuator/11
[564690142507250] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::The delivery has been complete. The delivery token is 1
[       Thread-3] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::[publish]::Send message to smartroom/4/monitor/actuator/12
[564690374520209] i.u.d.s.s.broker.MqttBroker             : [MqttBroker]::The delivery has been complete. The delivery token is 1
[       Thread-3] i.u.d.s.s.analyzer.Analyzer             : [Analyzer]::[analyzeSensorsValue] --- Run analyzer
[       Thread-3] i.u.d.s.smarthomeclima.planner.Planner  : [Planner]::[planning] --- Run planner
[       Thread-3] i.u.d.s.smarthomeclima.planner.Planner  : [Planner]::[planning] --- Save new pianification --- SmartRoom ---  1
[       Thread-3] i.u.d.s.smarthomeclima.planner.Planner  : [Planner]::[plannig] --- Deactivate pianification -- 2
[       Thread-3] i.u.d.s.smarthomeclima.planner.Planner  : [Planner]::[plannig] --- Deactivate pianification -- 3
[       Thread-3] i.u.d.s.s.executor.Executor             : [Executor]::[executor] --- Do the planned actions
[       Thread-3] i.u.d.s.s.executor.Executor             : [Executor]::[executor] --- Payload -- 2, 1, 1
```

Figure 11 - Application log

11. Conclusion

In this section we report the conclusions regarding the application developed and the behaviours adopted during the execution phase.

After carrying out the testing operation, leaving the system running, it was found that the **SmartHomeClima** responds highly efficiently to the needs of the application context. The system obtained is of the *Hybrid Initiative* type (see section 1); on the one hand it is capable of autonomously taking the appropriate initiatives for the current situation, on the other hand, the user also has the possibility of interacting with it, through the graphical interface, by manually changing the settings available.

In the reactive phase it was found that the system makes efficient choices based on the context analysed, respecting the times necessary for regular operation.

Analyzing the response times of the application, we can state that they are relatively short, even if the simulation launched sends data at intervals chosen by us, as we are able to monitor in real time both the situations present in each room, and the variation of the system states during the implementation of the different execution plans.

Therefore, the application would seem to meet all the requirements necessary for the creation of an autonomous system within the specified context.